Microcontroller

Presentation

By

Rajul Patkar

Microprocessor and Microcontroller

Microprocessors is a general purpose computer. Multi chip required for all the functions.

Microcontrollers is a true computer on a single chip. Logic functions, bit addressing, small cheap and fast.

Microprocessor and Microcontroller

Essential elements of any computer are:

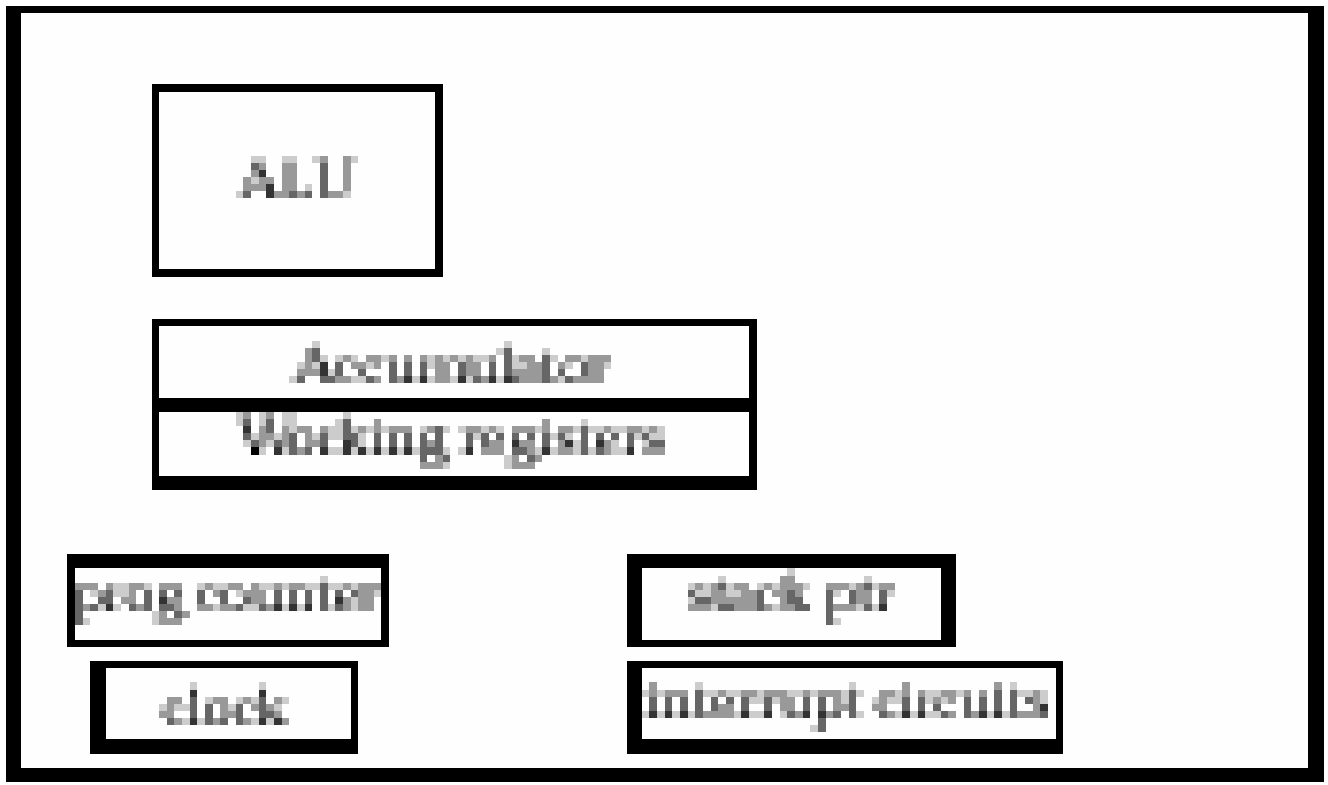
- **CPU**
- **Memory for both data and program**
- **I/O or Input Output system**

Microprocessor and Microcontroller

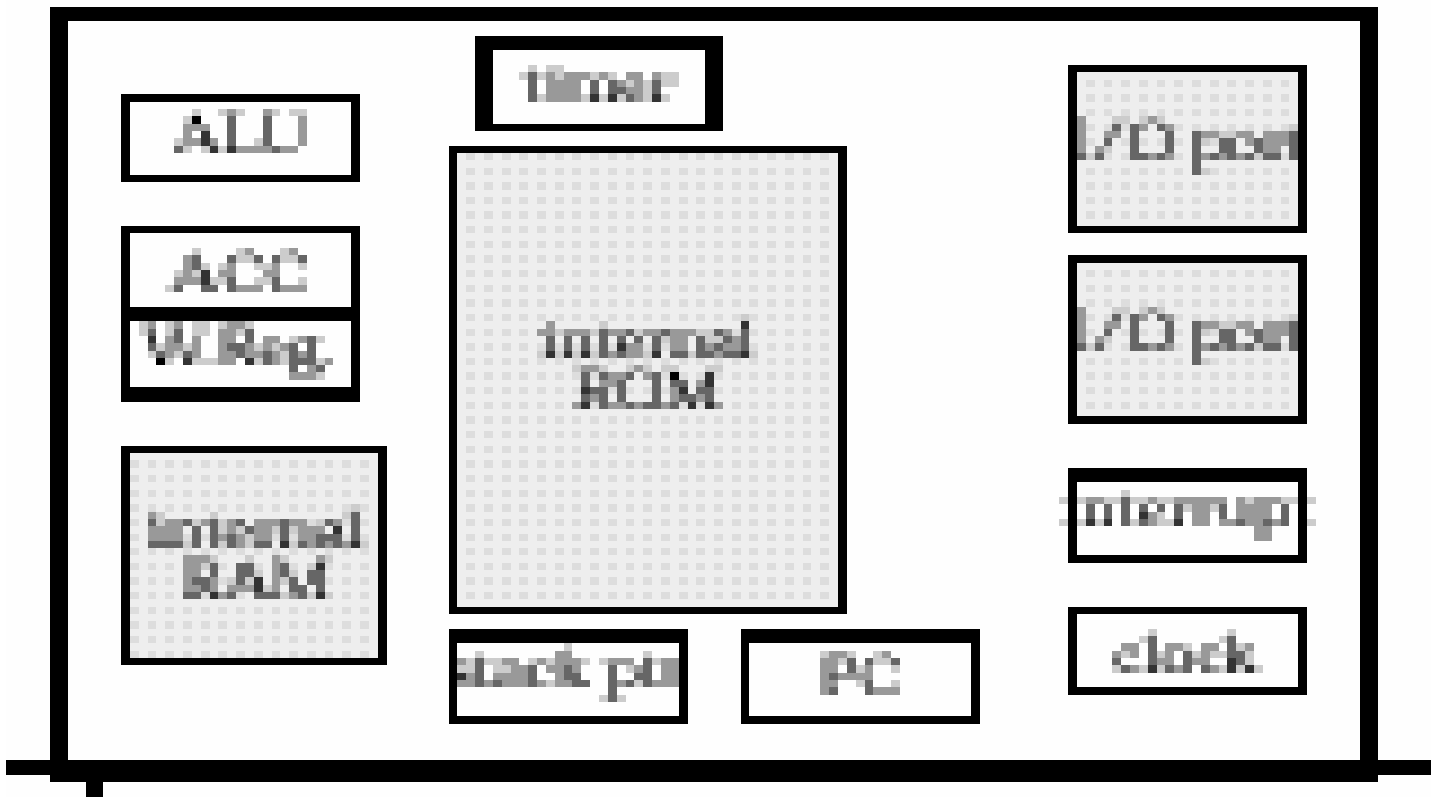
Various different architectures exist depending on the application

- Mini-computer: all 3 systems on a board
- Micro-controller: All systems on a single chip

Microprocessor



Microcontroller



A Microcontroller application

Intel 8051 Microcontroller

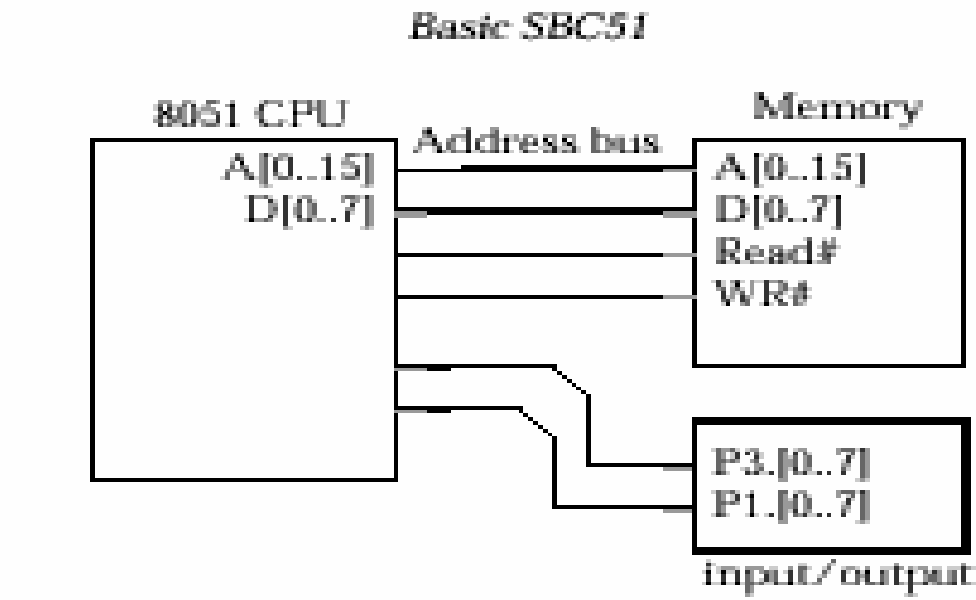
8051 is made by Intel. It is one of the most widely used microcontroller in the world.

- **A stand alone, high performance, single chip computer for control applications.**
- **Small, cheap and 40 pins.**
- **64K program memory.**
- **64K data memory.**

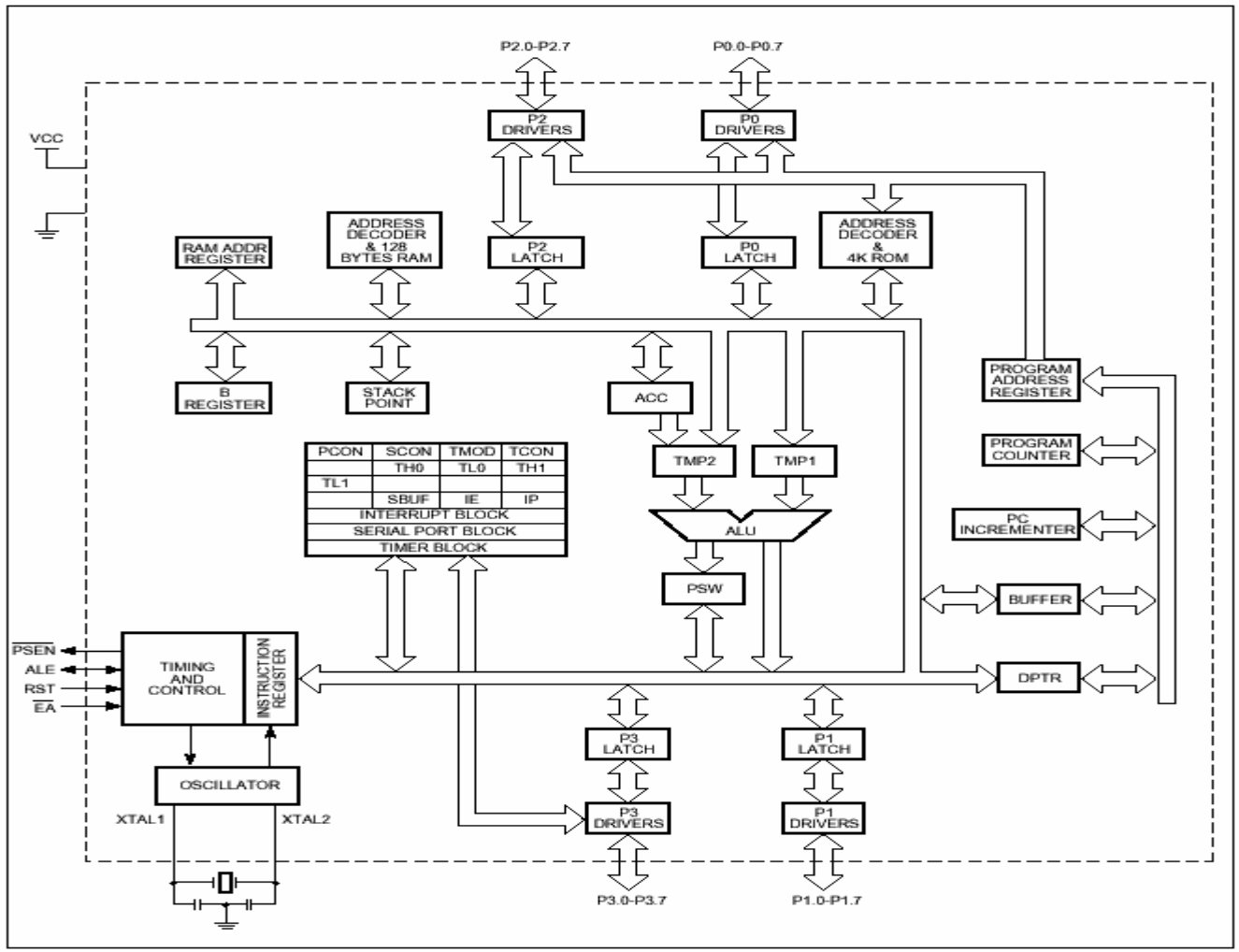
8051 Pin layout

8051 Controller

- A minimal 8051 based controller needs only the following components
- CPU
- Memory
- I/O



8051 Block Diagram



8051 Architecture

8051 architecture contains the following:

- 8 bit CPU with registers A and B
- 16 bit program counter(PC) and data pointer(DPTR)
- 8 bit program status word(PSW)
- 8 bit stack pointer
- Internal ROM of 0(8031) to 4K(8051)
- Internal RAM of 128 Bytes
  - 4 register banks 00-1f
  - 16 bytes(bit addressable) 20-2f
  - 80 bytes of general purpose data memory 30-7f

8051 Architecture

- 32 I/O pins arranged as four 8 bit ports (P0 – P3)
- 2 16-bit timer/counters: T0 and T1
- Full duplex serial data receiver/transmitter: SBUF
- Control registers: TCON, TMOD, SCON, PCON, IP and IE
- 2 external and 3 internal interrupt sources
- Oscillator and clock circuits

8051 Memory Types

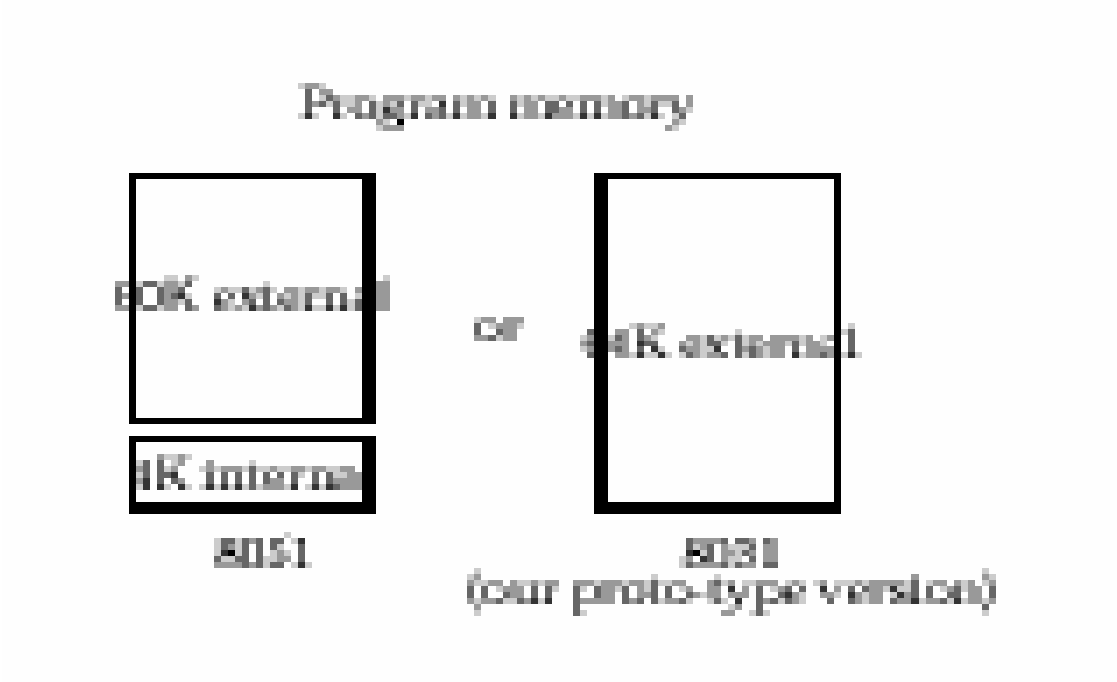
The 8051 has *two separate* memory blocks, for *data* and *program* . Since both blocks have the *same* address, this is called a Harvard architecture.

Separating the program memory from the data memory improves reliability since we cannot overwrite the program code. and allows us to use program *ROM* , or read only memory.

Internal circuitry accesses the correct memory based on the nature of the operation in progress.

Program Memory

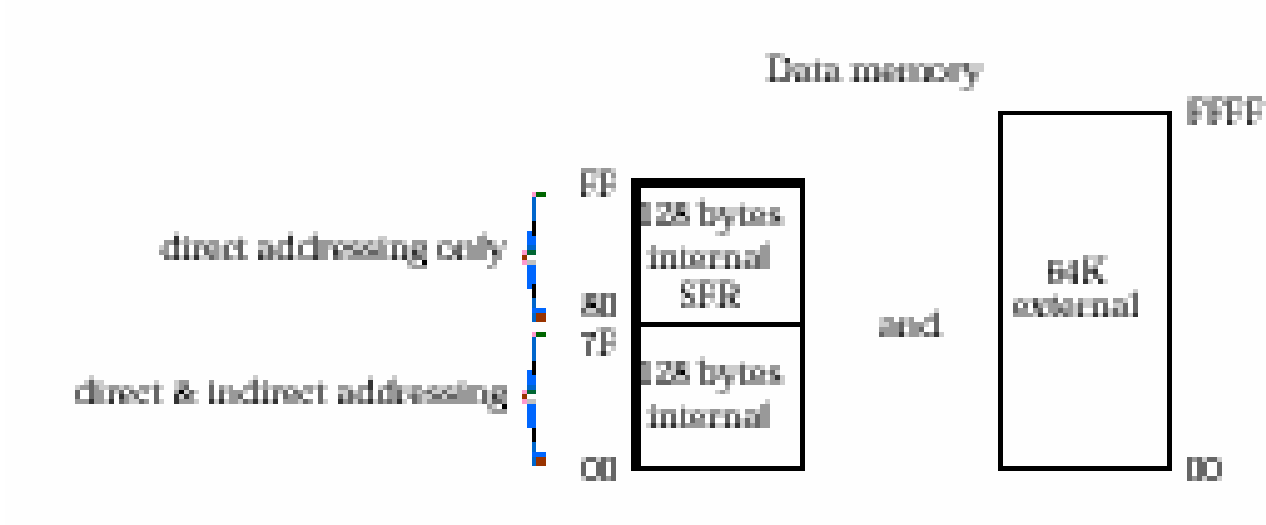
Program memory normally we have 4K on the chip (but not in the case of the 8031)



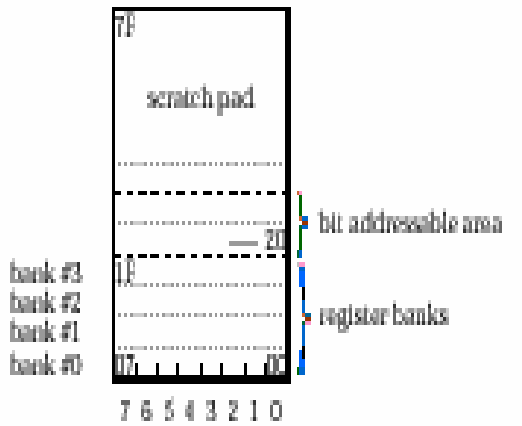
Data Memory

Data memory all 64K is off the chip except for (a miniscule 128 bytes).

Internal 256 bytes is divided into two parts: general scratch and the special function registers.



The 128 bytes in lower internal RAM have the following structure:



Only one of the 4 register banks can be active at any particular time Which one is active is given by flags in the PSW

Special Function Registers

The Special Function Registers (SFRs) contain memory locations that are used for special tasks. (They should not be used for general purpose tasks.)

Each SFR occupies internal RAM from 0x80 to 0xFF, (but some areas are empty!) They are 8 bits wide. Some examples are:

A register or accumulator is used for most ALU operations & external moves

B used for multiplication & division and can also be used for general purpose storage

PSW Program Status Word is a bit addressable register.

Two special 16- bit registers (double registers):

PC or program counter. This is not directly addressable, nor does it have a memory location. It is not part of SFR.

DPTR or data pointer. Is accessible as two 8- bit registers: DPL and DPH. DPTR doesn't have a single internal address; DPH and DPL are each assigned an address.

This is used to furnish memory addresses for internal and external code access and external data access.

Program Status Word

The PSW is the most important of the SFRs. It is a byte register which holds 8 bits that can addressed individually.

Each bit is referred to as a flag. Flags can be either set (1) or cleared (0). This is more efficient than using an entire byte memory location for a binary variable.

The PSW is divided up as follows:

7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	OV	–	P

SFRs

SFRs which are also bit addressable

A, B, IP, IE, TCON, SCON, PSW, P0, P1, P2, P3

Other SFRs

TMOD, THO, TLO, TH1, TL1, SBUF, PCON, SP, DPTR

Port Operations

Total 4 ports

Port 0 may serve as inputs, outputs, or as a low order

address and data bus for external memory.

Port 1 may be used as input/output port.

Port 2 may be used as input/output or high order

address byte.

Port 3 may be used as an input/output and for some

alternate function.

Port 3 Alternate Operations

Pin	Alternate use	SFR
P3.0-RXD	Serial data input	SBUF
P3.1-TXD	Serial data output	SBUF
P3.2-INT0~	Ext. Int. 0	TCON.1
P3.3-INT1~	Ext. Int. 1	TCON.3
P3.4-T0	Ext. Tim. 0	TMOD
P3.4-T1	Ext. Tim. 1	TMOD
P3.6-WR~	Ext. Mem write pulse	
P3.7-RD~	Ext. Mem Read pulse	

Port Operations

To read and write from port:

MOV A, P0 or MOV A,80h

- This copies data from port 0 pins to register A.

MOV P1, #0a5h or MOV 90h,#0a5h

- This moves a constant number into port1.

Moving data to a port changes the port latch, moving data from a port gets data from the port pins.

Memory Mapped Port Operations

Store the address of the memory mapped port into the DPTR register, and indirectly address this through Acc.

- Reading from a port

```
MOV DPTR, #0F012h; address to read digital
    input
MOVX A, @DPTR ; read the values into Acc
```

- Writing to a port

```
MOV DPTR, #0F011h; address of LED outputs
MOVX @DPTR, A ; write this value back out
    again
```

Accessing external memory

Storage problems?

What happens if a program is larger than 4096 bytes of code?

We have the 'ROMless' 8031 version, we must use the movx

(move external) to access the data block, and movc to fetch

from code memory.

We have two separate read signals,

RD~, (read) and PSEN~, (program send enable).

Prototyping 8051 Design

For many small applications where we might want to Change the ROM code, we would like to use *external EPROM* , which means:

- **We have lost the use of ports 0 and ports 2 since they must be used to interface with external ROM and not available for I/ O.**
- **We may need external RAM (since the 128 bytes may not be enough for our application), which means that we have lost port 3.6, WR~, and 3.7, RD~**
- **Any serial communication requires 3.0 (RXD) and 3.1(TXD)**

Loss of ports leaves us with:

- 1. All of port #1, and
- 2. Port P3.2 – P3.5

for general purpose I/ O and external interrupts and timing inputs.

What is to be done for the lost ports?

Programmable Logic Devices

The PROM chip is an example of a Programmable Logic Device PLD. Once programmed, it will retain the program even when the power is turned off.

One- time programmable devices uses tiny fuses which are burnt or blown (or we can use antifuses which have the same end function.

Eraseable Programmable read Only memory (EPROMS) can be re- programmed. This uses a modified MOS transistor with a floating gate that when uncharged does not effect the normal operation. However if it is subjected to (high) +12V, then a charge will move into the 2nd transistor which is stable (will last a decade). Reprogram by subjecting the cells to UV light via a ceramic window on top of the package. Wiping time takes about 20 minutes.

EEPROM or Electrically erasable proms are about 2.5 times larger than eproms, and are quicker to clear.

FLASH proms are much quicker to erase (hence the name), and can be reprogrammed *while still on the circuit board* . Suitable for easy upgrading by the public of ROM chips in Modems and PCs.

Interrupts

Interrupts are just *special* subroutines that may (or may not) be called explicitly.

If conditions are "right", when an interrupt occurs, then the processor will stop what it is doing, and jump to a specific place in memory (decided by the Intel 8051 designers) hooked by that particular interrupt.

It is up to the programmer to make sure that you supply a sensible further course of action. This is called the interrupt handler routine or *interrupt service routine* , ISR.

Interrupts

Interrupt programming

Signals or conditions generated *external* to the main program

– External events such as a change in a logic value

– Overflow of a counter

– Arrival of data at the serial port.

Interrupts can be enabled or disabled by the programmer. (Although some interrupts are non- maskable.)

Types of Interrupts

Five interrupts are provided on 8051.

3 are generated by internal operations.

Generated by internal timer/counter

- **Timer flag 0 – TF0**
- **Timer flag 1 – TF1**

  Indicates that a character has been received or the buffer is empty and a character can be transmitted
- **Serial port interrupt (RI or TI)**

2 are triggered by external signals

- **INT0~**
- **INT1~**

Interrupts

All interrupt functions are under the control of the program. The programmer can alter the bits of IE, IP and TCON register.

Each interrupt forces the processor to jump at the interrupt location in the memory.

The interrupted program must resume operation at the instruction where the interrupt took place.

Program resumption is done by storing the interrupted PC address on to stack.

RETI instruction at the end of ISR will restore the PC address.

Timers and Counters

The 8051 comes equipped with two timers, both of which may be controlled, set, read, and configured individually. The 8051 timers have three general functions:

1) Keeping time and/or calculating the amount of time between events,
2) Counting the events themselves, or
3) Generating baud rates for the serial port.

Timers and Counters

Could use software techniques, but this keeps the processor occupied. Better to use interrupts & the two 16- bit *count- up* timers. Can either be programmed to:

1. count internal - acting as timer

2. count external - acting as counter

All counter action is controlled by the TMOD (timer mode register) and the TCON (timer/ counter control register).

Timers and Counters

- **TCON** Timer control SFR contains timer 1& 2 overflow flags, external interrupt flags, timer control bits, falling edge/ Low level selector bit etc.

- **TMOD** timer mode SFR is two four- bit registers (timer #1, timer #0). Select timer/ counter & various modes.)

Timers and Counters

Applications: (counter)

Say we want to count a specified number of events (clock pulses or external events), then

1. **Store the start number in the counter. (Value maxcount-desired count+ 1)**
2. **Counter automatically increments (in the background)**
3. **When it *rolls over* to zero, it will set the timer flag.**
4. **Test the flag in the program, *or generate an interrupt* .**

Timers and Counters

Applications: (Timer)

Timing configures the counter to count the internal clock frequency/ 12. (E. g. if f_c = 6:0Mhz, then the timer clock will have a frequency of 500kHz.)

To configure as a timer:

1. Clear C/T~ bit in TMOD (Count internal frequency)

2. Set TRx in the TCON (timer run) *and* the gate bit in the TMOD must be 0, or the external pin INTx~ must be 1.

3. Select one of 4 modes.

Timing Subroutines

In microcontroller applications we need to wait a specified time, and so we need to have programmable time delays.

In all cases your application will be written for a specific clock frequency, say 12MHz, 16MHz, 11.0592 MHz etc.)

If you subsequently change the clock frequency, you will need to update your code, & possibly do some timing tests.

Timing Subroutines

The timing options are:

1. Pure software time delays (wastefull)
2. Software polled timer (still wastefull, but OK in non- critical apps)
3. Pure hardware using interrupts (accurate & preferred method)